BLM6112
Advanced Computer Architecture
Fundamentals of Quantitative Design and Analysis

Prof. Dr. Nizamettin AYDIN

naydin@yildiz.edu.tr

http://www3.yildiz.edu.tr/~naydin

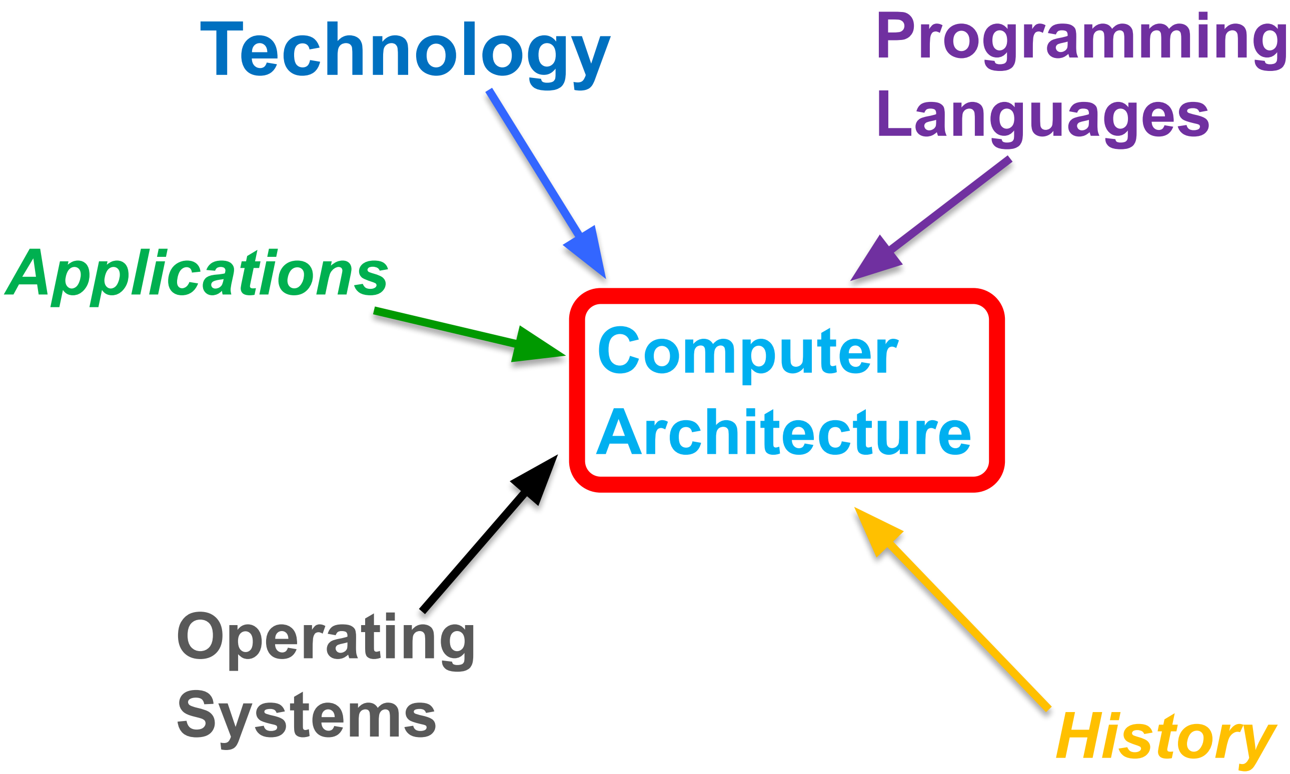
Performance Metrics

- Objectives
  - How can we meaningfully measure and compare computer performance?
  - Understand why program performance varies
    - Understand how applications and the compiler impact performance
    - Understand how CPU impacts performance
    - What trade-offs are involved in designing a CPU?
  - Purchasing perspective vs design perspective

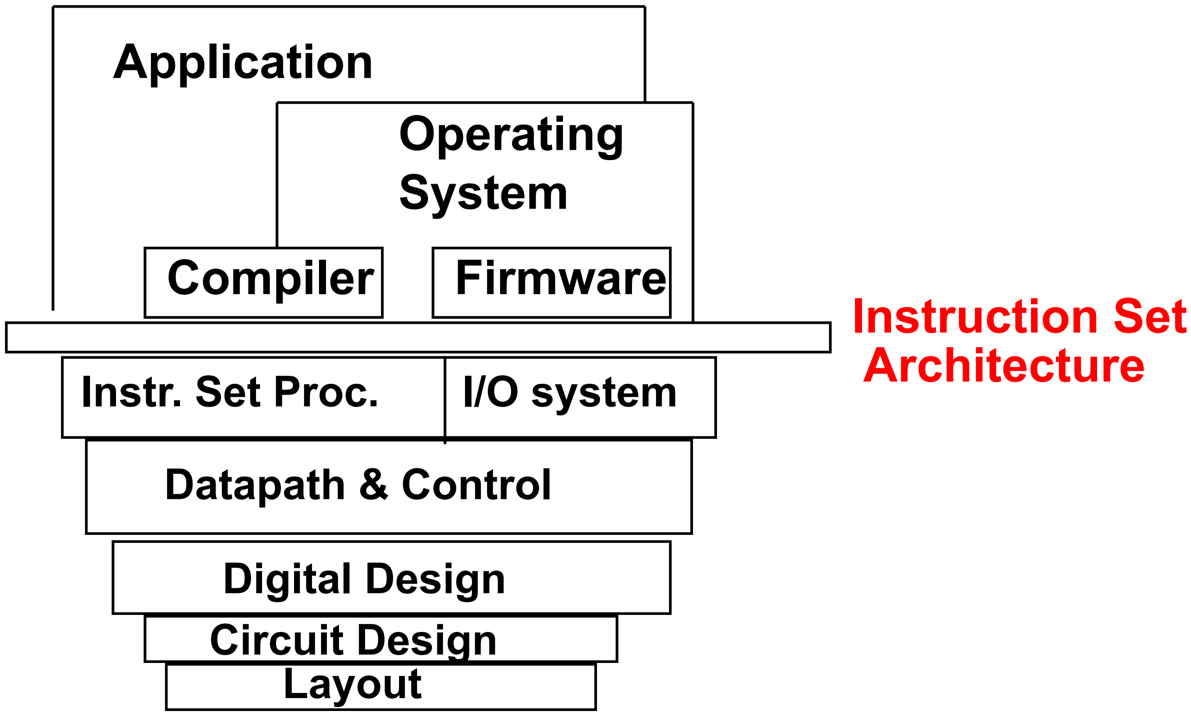
Outline

- Latency, delay, time
- Throughput
- Cost
- Power
- Energy
- Reliability

Forces on Computer Architecture



What is Computer Architecture?

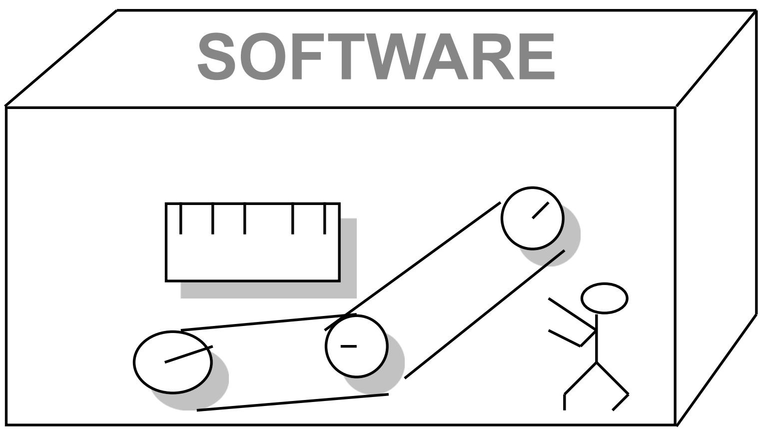


- Coordination of many *levels of abstraction*
  - Under a rapidly changing set of forces
- Design, Measurement, and Evaluation

Computer Architecture is

The attributes of a [computing] system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

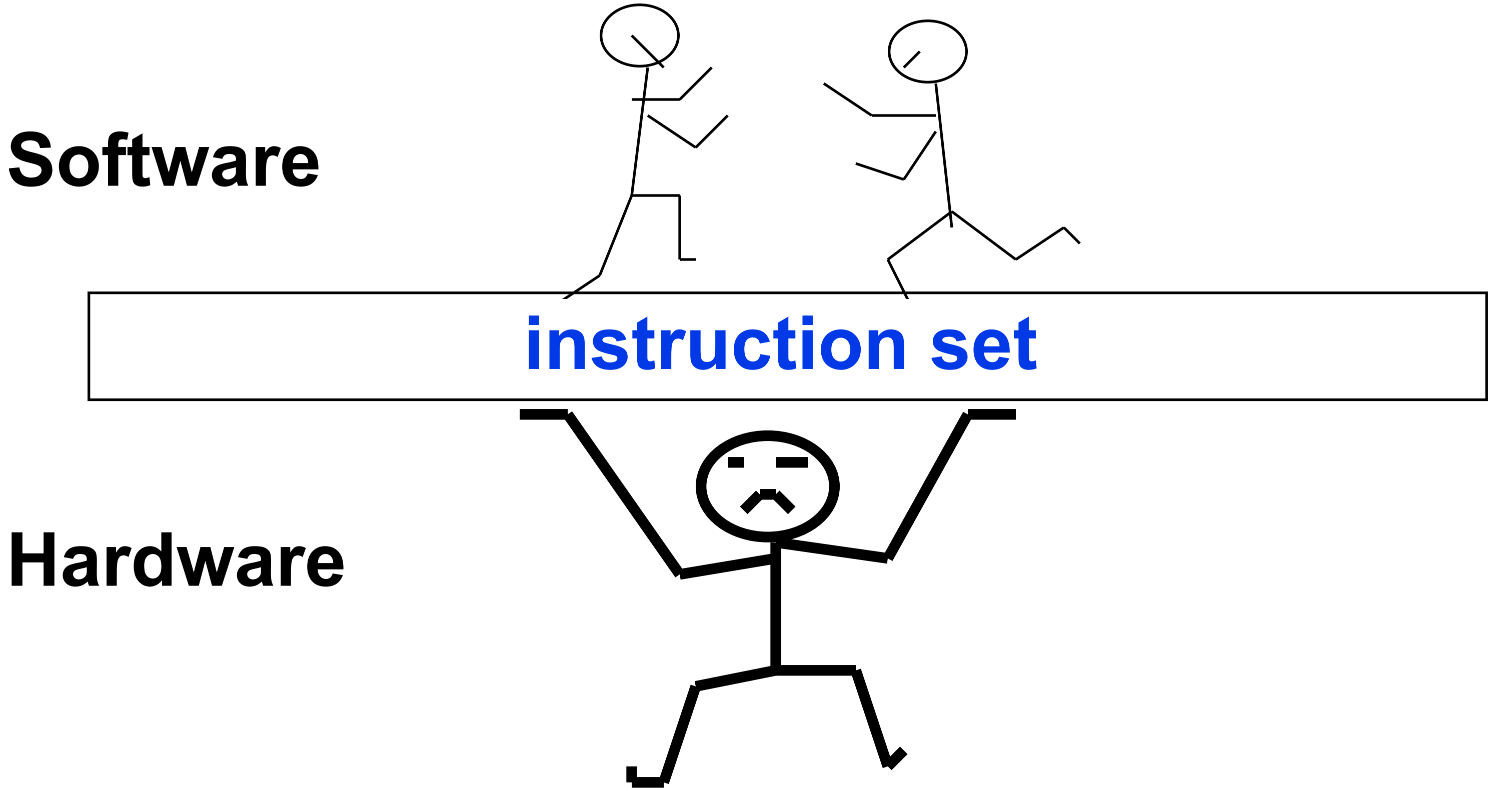
Amdahl, Blaaw, and Brooks, 1964



A Changing Definition

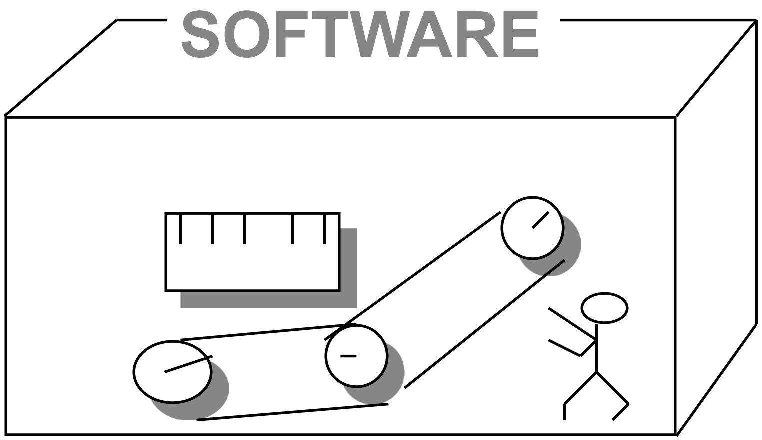
- 1950s to 1960s: Computer Architecture Course
  - Computer Arithmetic
- 1970s to mid 1980s: Computer Architecture Course
  - Instruction Set Design, especially ISA appropriate for compilers
- 1990s: Computer Architecture Course
  - Design of CPU, memory system, I/O system, Multiprocessors

The Instruction Set: a Critical Interface



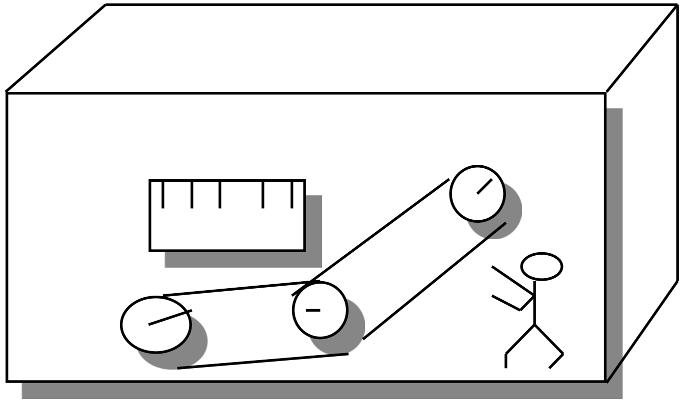
Instruction Set Architecture

- Organization of Programmable Storage
- Data Types & Data Structures Encodings & Representations
- Instruction Formats
- Instruction (or Operation Code) Set
- Modes of Addressing and Accessing Data Items and Instructions
- Exceptional Conditions



Computer Organization

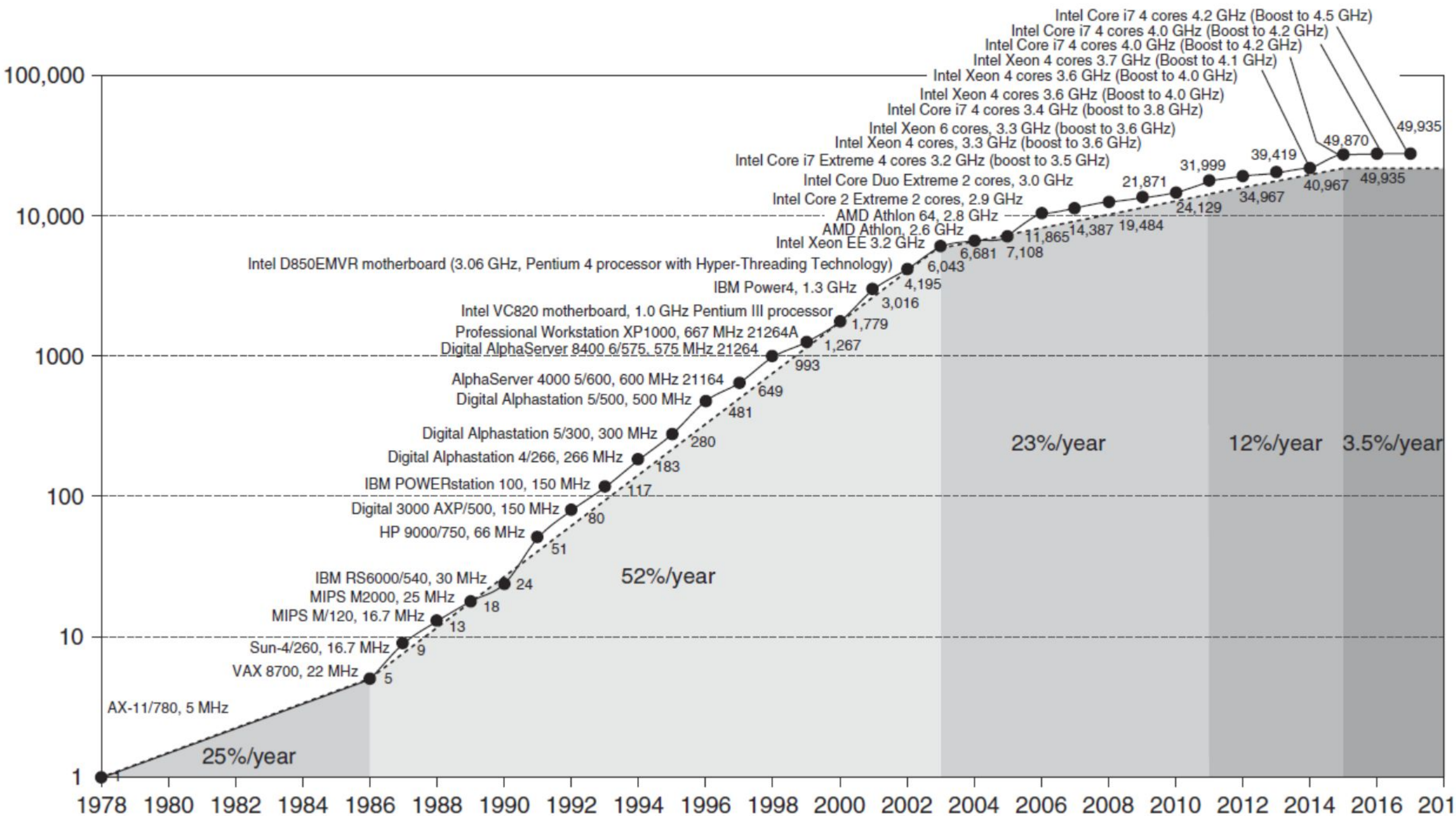
- Capabilities & Performance Characteristics of Principal Functional Units
  - (e.g., Registers, ALU, Shifters, Logic Units, ...)
- Ways in which these components are interconnected
- Information flows between components
- Logic and means by which such information flow is controlled.
- Choreography of FUs to realize the ISA
- Register Transfer Level (RTL) hardware design



Computer Technology

- Performance improvements:
  - Improvements in semiconductor technology
    - Feature size, clock speed
  - Improvements in computer architectures
    - Enabled by HLL compilers, UNIX
    - Lead to RISC architectures
  - Together have enabled:
    - Lightweight computers
    - Productivity-based managed/interpreted programming languages

Growth in processor performance over 40 years



Current Trends in Architecture

- Cannot continue to leverage Instruction-Level Parallelism (ILP)
  - Single processor performance improvement ended in 2003
- New models for performance:
  - Data-Level Parallelism (DLP)
  - Thread-Level Parallelism (TLP)
  - Request-Level Parallelism (RLP)
- These require explicit restructuring of the application

Classes of Computers

- Personal Mobile Device (PMD)
  - e.g. start phones, tablet computers
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers
  - Used for "Software as a Service (SaaS)"
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Internet of Things/Embedded Computers
  - Emphasis: price

A summary of the five mainstream computing classes and their system characteristics

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Internet of things/ embedded
Price of system	$100–$1000	$300–$2500	$5000–$10,000,000	$100,000–$200,000,000	$10–$100,000
Price of microprocessor	$10–$100	$50–$500	$200–$2000	$50–$250	$0.01–$100
Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

- Sales in 2015 included about
  - 1.6 billion PMDs (90% cell phones),
  - 275 million desktop PCs,
  - 15 million servers.
  - 19 billion embedded processors.
- In total, 14.8 billion ARM-technology-based chips were shipped in 2015

Parallelism

- Parallelism at multiple levels is now the driving force of computer design across all four classes of computers,
  - with energy and cost being the primary constraints.
- Classes of parallelism in applications:
  - Data-Level Parallelism (DLP)
    - arises because there are many data items that can be operated on at the same time.
  - Task-Level Parallelism (TLP)
    - arises because tasks of work are created that can operate independently and largely in parallel.

Parallelism

- Computer hardware in turn can exploit these two kinds of application parallelism in four major ways:
  - Instruction-Level Parallelism (ILP)
    - exploits data-level parallelism at modest levels with compiler help using ideas like pipelining and at medium levels using ideas like speculative execution.
  - Vector architectures/Graphic Processor Units (GPUs) and multimedia instruction sets
    - exploit data-level parallelism by applying a single instruction to a collection of data in parallel.

Parallelism

- Thread-Level Parallelism (TLP)
  - exploits either data-level parallelism or task-level parallelism in a tightly coupled hardware model that allows for interaction between parallel threads.
- Request-Level Parallelism (RLP)
  - exploits parallelism among largely decoupled tasks specified by the programmer or the operating system.

- When Flynn (1966) studied the parallel computing efforts in the 1960s, he found a simple classification whose abbreviations we still use today.
  - He looked at the parallelism in the instruction and data streams called for by the instructions at the most constrained component of the multiprocessor and placed all computers in one of four categories:

Flynn’s Taxonomy

- Single instruction stream, single data stream (SISD)
  - the uniprocessor
  - it can exploit ILP
- Single instruction stream, multiple data streams (SIMD)
  - The same instruction is executed by multiple processors using different data streams
  - exploit DLP by applying the same operations to multiple items of data in parallel
    - Vector architectures
    - Multimedia extensions
    - Graphics processor units

Flynn's Taxonomy

- Multiple instruction streams, single data stream (MISD)
  - No commercial implementation
- Multiple instruction streams, multiple data streams (MIMD)
  - Each processor fetches its own instructions and operates on its own data, and it targets TLP
  - can also exploit DLP
  - more flexible than SIMD and thus more generally applicable, but it is inherently more expensive than SIMD
    - Tightly-coupled MIMD
    - Loosely-coupled MIMD

Defining Computer Architecture

- "Old" view of computer architecture:
  - Instruction Set Architecture (ISA) design
  - i.e. decisions regarding:
    - registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding

- "Real" computer architecture:
  - Specific requirements of the target machine
  - Design to maximize performance within constraints:
    - cost, power, and availability
  - Includes ISA, microarchitecture, hardware

Instruction Set Architecture

- Class of ISA
  - General-purpose registers
  - Register-memory vs load-store
- RISC-V registers
  - 32 g.p., 32 f.p.

Register	Name	Use	Saver
x0	**zero**	**constant 0**	**n/a**
x1	**ra**	**return addr**	**caller**
x2	**sp**	**stack ptr**	**callee**
x3	**gp**	**gbl ptr**	
x4	**tp**	**thread ptr**	
x5-x7	**t0-t2**	**temporarie s**	**caller**
x8	**s0/fp**	**saved/ frame ptr**	**callee**

Register	**Name**	**Use**	**Saver**
x9	**s1**	**Saved**	**callee**
x10-x17	**a0-a7**	**Arguments**	**caller**
x18-x27	**s2-s11**	**Saved**	**callee**
x28-x31	**t3-t6**	**Temporaries**	**caller**
f0-f7	**ft0-ft7**	**FP temps**	**caller**
f8-f9	**fs0-fs1**	**FP saved**	**callee**
f10-f17	**fa0-fa7**	**FP arguments**	**callee**
f18-f27	**fs2-fs21**	**FP saved**	**callee**
f28-f31	**ft8-ft11**	**FP temps**	**caller**

Instruction Set Architecture

- Memory addressing
  - RISC-V: byte addressed, aligned accesses faster
- Addressing modes
  - RISC-V: Register, immediate, displacement (base+offset)
  - Other examples: autoincrement, indexed, PC-relative
- Types and size of operands
  - RISC-V: 8-bit, 32-bit, 64-bit

Instruction Set Architecture

- Operations
  - RISC-V: data transfer, arithmetic, logical, control, floating point
    - See Fig. 1.5 in text, or The RISC-V Instruction Set Manual
- Control flow instructions
  - Use content of registers (RISC-V) vs. status bits (x86, ARMv7, ARMv8)
  - Return address in register (RISC-V, ARMv7, ARMv8) vs. on stack (x86)
- Encoding
  - Fixed (RISC-V, ARMv7/v8 except compact instruction set) vs. variable length (x86)

Trends in Technology

- Integrated circuit technology (Moore's Law)
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year

- DRAM capacity: 25-40%/year (slowing)
  - 8 Gb (2014), 16 Gb (2019), possibly no 32 Gb

- Flash capacity: 50-60%/year
  - 8-10X cheaper/bit than DRAM

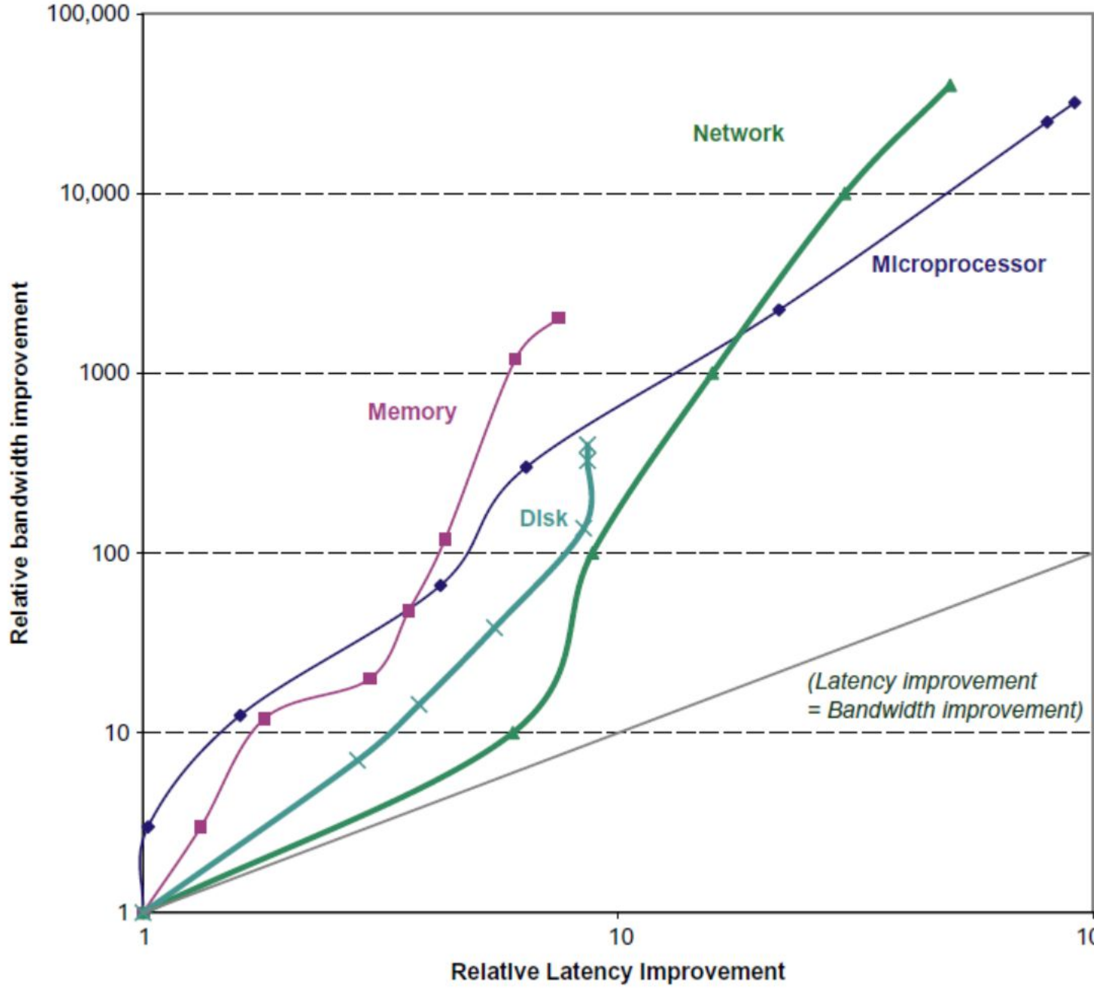
- Magnetic disk capacity: recently slowed to 5%/year
  - Density increases may no longer be possible,
    - may be increase from 7 to 9 platters
  - 8-10X cheaper/bit then Flash
  - 200-300X cheaper/bit than DRAM

Bandwidth and Latency

- Bandwidth or throughput
  - Total work done in a given time
  - 32,000-40,000X improvement for processors
  - 300-1200X improvement for memory and disks

- Latency or response time
  - Time between start and completion of an event
  - 50-90X improvement for processors
  - 6-8X improvement for memory and disks

Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

Transistors and Wires

- Feature size
  - Minimum size of transistor or wire in *x* or *y* dimension
    - 10 µm in 1971
    - 0.011 µm in 2017
    - 0.003 µm in 2021
  - Transistor performance scales linearly
    - Wire delay does not improve with feature size!
  - Integration density scales quadratically

Power and Energy

- Problem:
  - Get power in, get power out
- Thermal Design Power (TDP)
  - Characterizes sustained power consumption
  - Used as target for power supply and cooling system
  - Lower than peak power (1.5X higher), higher than average power consumption
- Clock rate can be reduced dynamically to limit power consumption
- Energy per task is often a better measurement

Dynamic Energy and Power

- Dynamic energy

$$= \frac{Capacitive\ Load \times Voltage^2}{2}$$

  – Transistor switch from 0 ➔ 1 or 1 ➔ 0

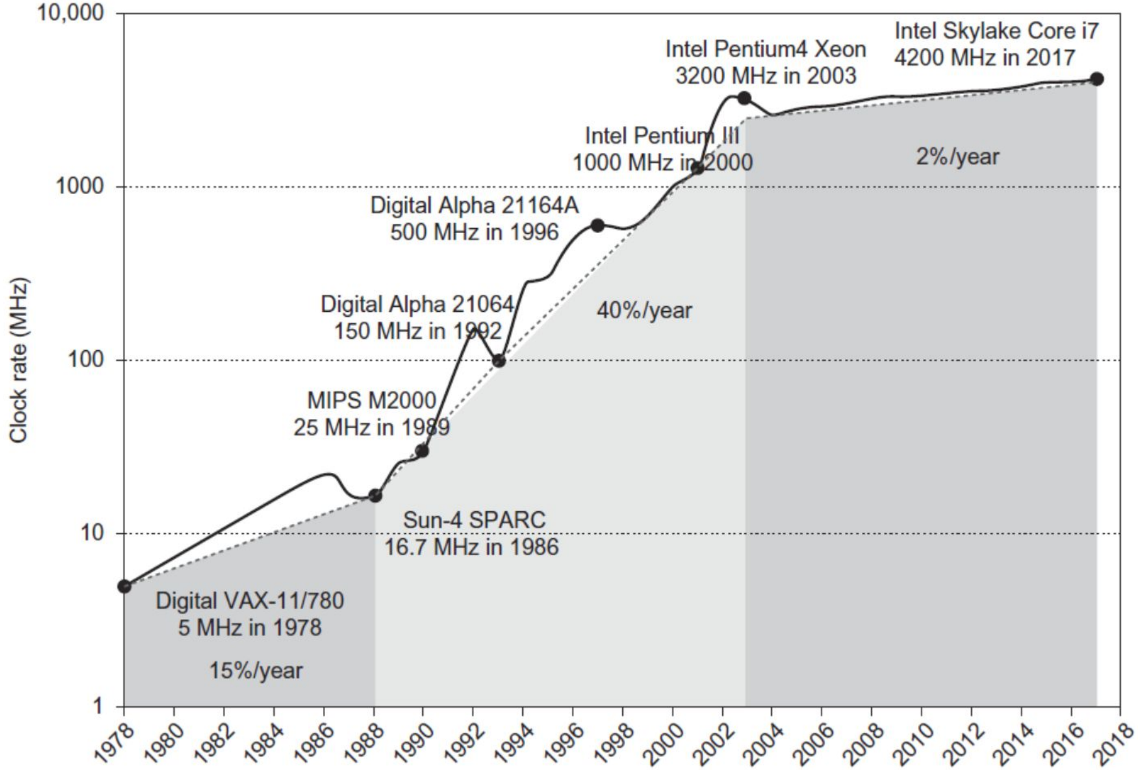
- Dynamic power

$$= \frac{Capacitive\ Load \times Voltage^2 \times Frequency\ switched}{2}$$

- Reducing clock rate reduces power, not energy

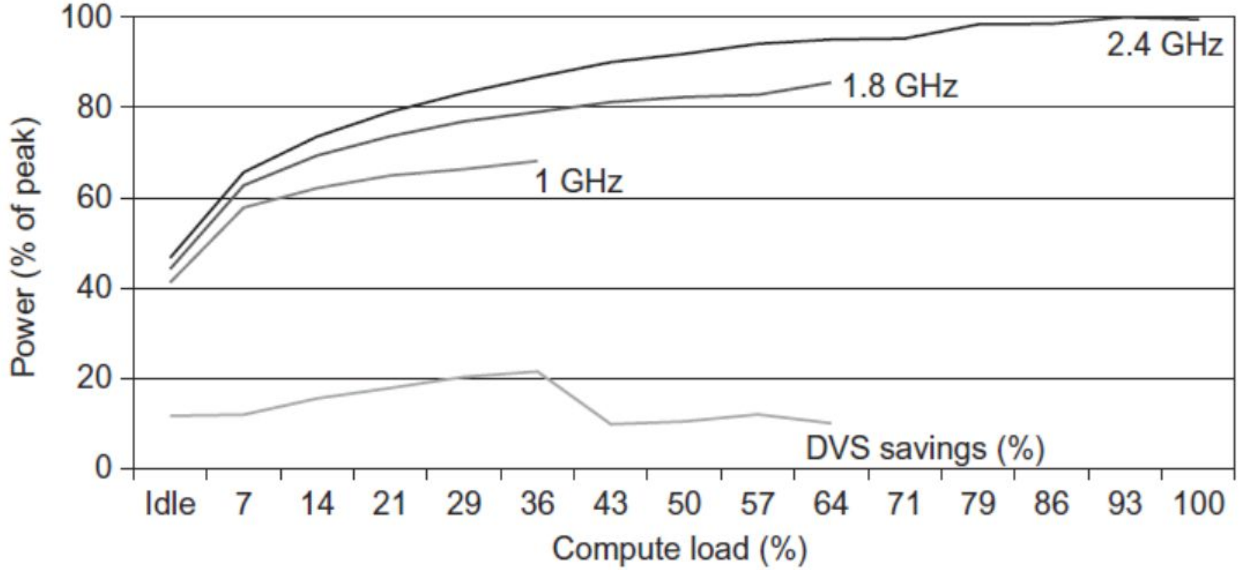
Power

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Core i7 consumes 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air



Reducing Power

- Techniques for reducing power:
  - Do nothing well
  - Dynamic Voltage-Frequency Scaling



  - Low power state for DRAM, disks
  - Overclocking, turning off cores

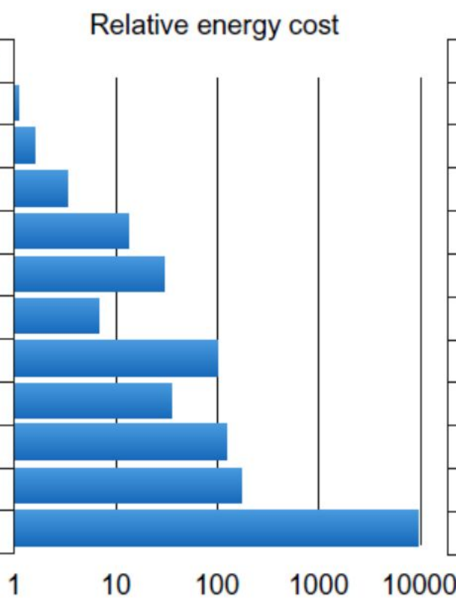
Static Power

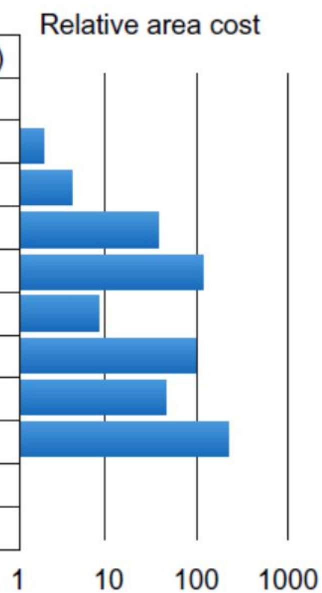
- Static power consumption

$$= Current_{static} \times Voltage$$

  - 25-50% of total power
  - Scales with number of transistors
- To reduce:
  - power gating

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FB Add	0.4	1360
32b FB Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FB Mult	1.1	1640
32b FB Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





Trends in Cost

- Cost driven down by learning curve
  - Yield
- DRAM:
  - price closely tracks cost
- Microprocessors:
  - price depends on volume
    - 10% less for each doubling of volume

Integrated Circuit Cost

- Integrated circuit

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer x Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

- Bose-Einstein formula:

$$\text{Die yield} = \text{Wafer yield} \times 1/(1 + \text{Defects per unit area} \times \text{Die area})^N$$

- Defects per unit area = 0.016-0.057 defects per square cm (2010)
- N = process-complexity factor = 11.5-15.5 (40 nm, 2010)

Dependability

- Module reliability
  - Mean time to failure (MTTF)
  - Mean time to repair (MTTR)
  - Mean time between failures (MTBF) = MTTF + MTTR
  - Availability = MTTF / MTBF

Measuring Performance

- Typical performance metrics:
  - Response time
  - Throughput

- Speedup of X relative to Y
  - Execution time$_Y$ / Execution time$_X$

- Execution time
  - Wall clock time: includes all system overheads
  - CPU time: only computation time

- Benchmarks
  - Kernels (e.g. matrix multiply)
  - Toy programs (e.g. sorting)
  - Synthetic benchmarks (e.g. Dhrystone)
  - Benchmark suites (e.g. SPEC06fp, TPC-C)

Basic Performance Metrics

- Latency, delay, time
  - Lower is better
    - Complete a task as soon as possible
  - Measured in sec, µs, ns
- Throughput (bandwith)
  - Higher is better
    - Complete as many tasks per time as possible
  - Measured in bytes/sec, instructions/sec
- Cost
  - Lower is better
    - Complete tasks for as little money as possible
  - Measured in $, TL, etc.

Basic Performance Metrics

- Power
  - Lower is better
    - Complete tasks while dissipating as few joules/sec as possible
- Energy
  - Lower is better
    - Complete tasks using as few joules as possible
  - Measured in Joules, Joules/instruction
- Reliability
  - Higher is better
    - Complete tasks with low probability of failure
  - Measured in Mean time to failure (MTTF)
    - MTTF: the average time until a failure occurs

Bandwidth and Latency

- Bandwidth or throughput
  - Total work done in a given time
    - 32,000-40,000X improvement for processors
    - 300-1200X improvement for memory and disks
- Latency or response time
  - Time between start and completion of an event
    - 50-90X improvement for processors
    - 6-8X improvement for memory and disks

Latency vs Throughput

Plane	Istanbul to Madrid (hours)	Speed	Passengers	Throughput Passenger/Hour
Aircraft 1	4 hrs	900 km /hr	400	100
Aircraft 2	4.8 hrs	750 km/hr	600	125

- Madrid to Istanbul is about 3600 km
- Time:
  - Aircraft 1 is faster than Aircraft 2
    - 900/750 = 1.2 times or 20% faster
- Throughput:
  - Aircraft 2 has a higher throughput
    - (750*600)/(900*400) = 1.25 times the throughput or 25% more throughput

Response Time vs Throughput

- Response time (latency)
  - the time between the start and the completion of a task
    - Important to individual users (passengers)
- Throughput (bandwidth)
  - the total amount of work done in a given time
    - Important to data center managers (airline)

- Different performance metrics are required
  - to benchmark embedded and desktop computers,
    - which are more focused on response time,
  - to benchmark servers,
    - which are more focused on throughput

Principles of Computer Design

- Take Advantage of Parallelism
  - e.g. multiple processors, disks, memory banks, pipelining, multiple functional units

- Principle of Locality
  - Reuse of data and instructions

- Focus on the Common Case
  - Amdahl's Law

$$Execution\ time_{new} = Execution\ time_{old} \times \left((1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)$$

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Principles of Computer Design

- The Processor Performance Equation

$$CPU\ time = CPU\ clock\ cycles\ for\ a\ program \times Clock\ cycle\ time$$

$$CPU\ time = \frac{CPU\ clock\ cycles\ for\ a\ program}{Clock\ rate}$$

$$CPI = \frac{CPU\ clock\ cycles\ for\ a\ program}{Instruction\ count}$$

$$CPU\ time = Instruction\ count \times Cycles\ per\ instruction \times Clock\ cycle\ time$$

$$\frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Seconds}{Clock\ cycle} = \frac{Seconds}{Program} = CPU\ time$$

Principles of Computer Design

- Different instruction types having different CPIs

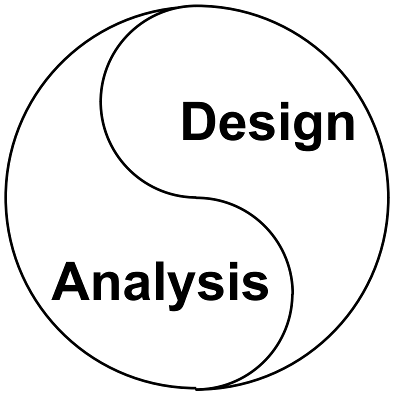
$$CPU\ clock\ cycles = \sum_{i=1}^{n} IC_i \times CPI_i$$

$$CPU\ time = \left(\sum_{i=1}^{n} IC_i \times CPI_i\right) \times Clock\ cycle\ time$$

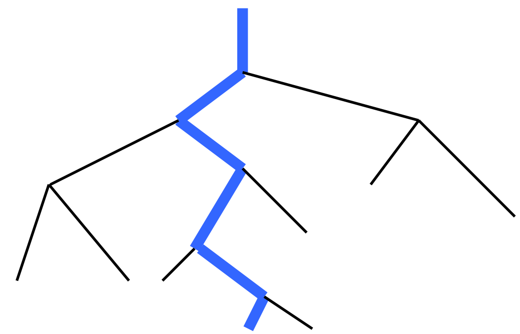
What is Performance?

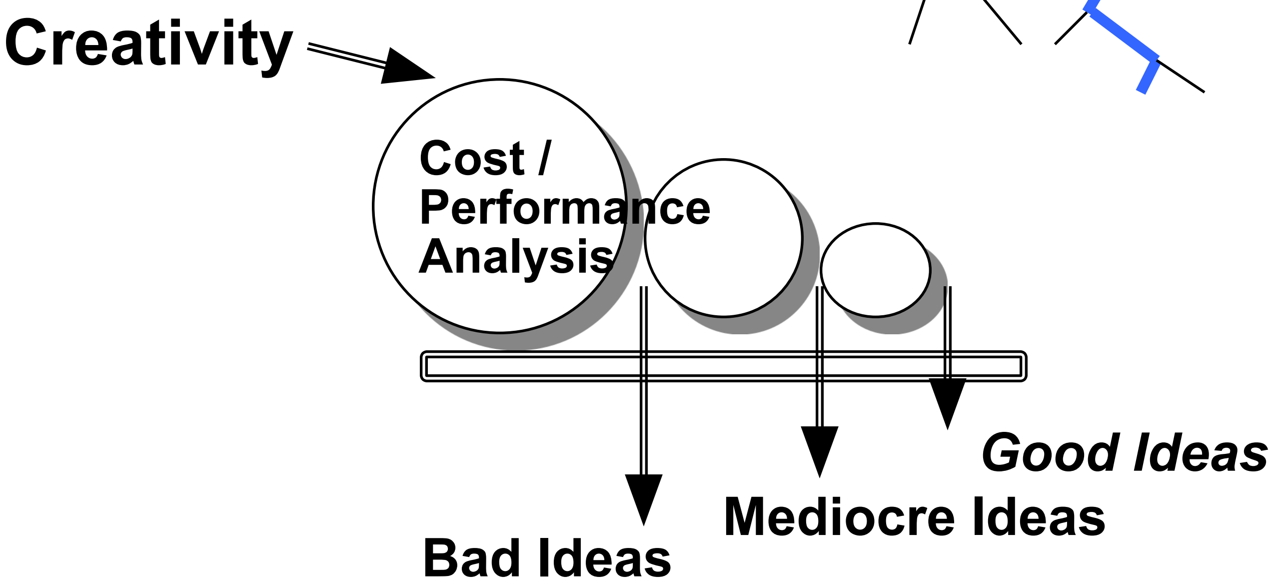
- Purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best performance / cost ?
- Design perspective
  - faced with design options, which has the
    - best performance improvement ?
    - least cost ?
    - best performance / cost ?
- Both require
  - basis for comparison
  - metric for evaluation
- Our goal is to understand cost & performance implications of architectural choices

Measurement and Evaluation



Architecture is an iterative process
-- searching the space of possible designs
-- at all levels of computer systems





Levels of Representation

High Level Language Program

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

Compiler

Assembly Language Program

```
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
```

Assembler

Machine Language Program

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Machine Interpretation

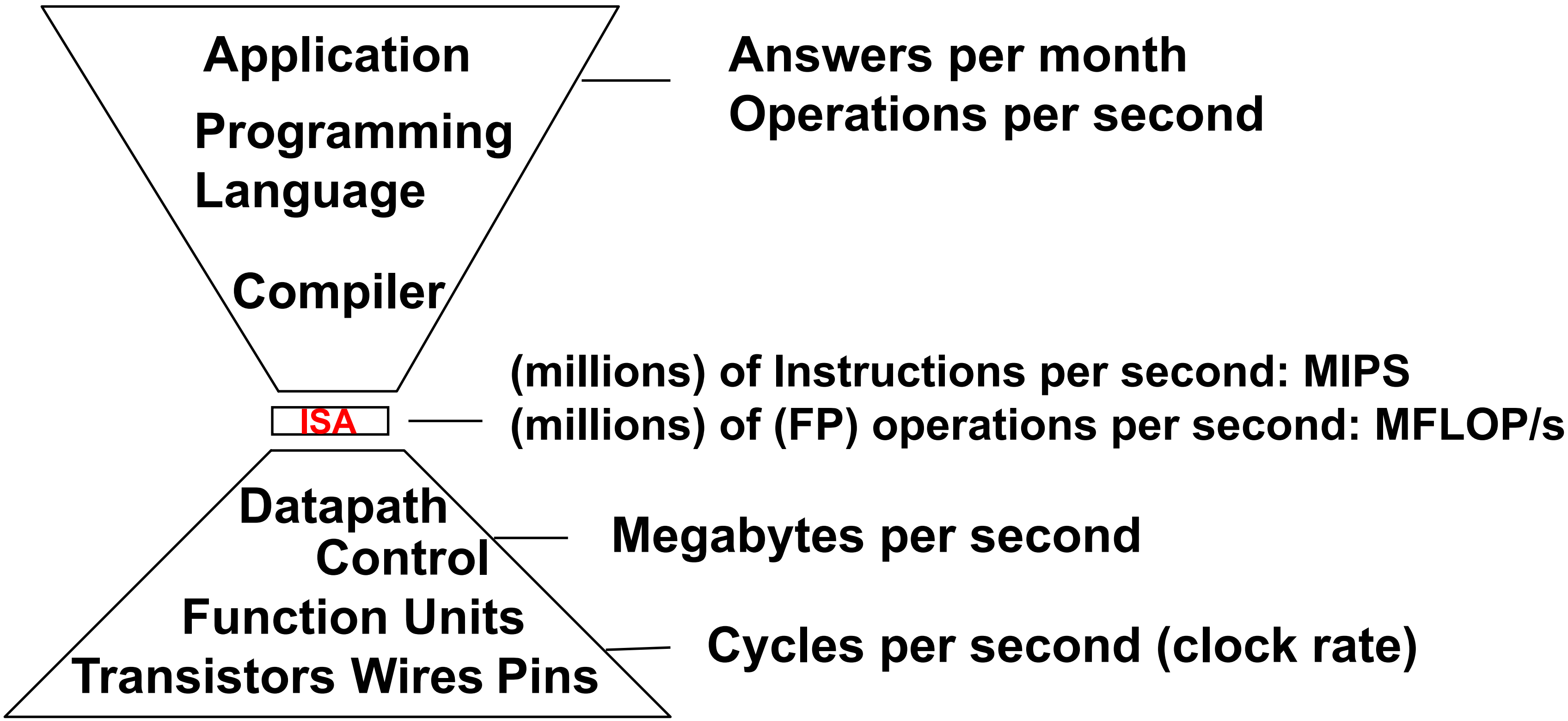
Control Signal Specification

ALUOP[0:3] <= InstReg[9:11] & MASK

°

°

Metrics of Performance



Each metric has a place and a purpose, and each can be misused

Basis of Evaluation

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run/measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	• less representative
• easy to run, early in design cycle	Small "Kernel" Benchmarks	• easy to "fool"
• identify peak capability and potential bottlenecks	Microbenchmarks	• "peak" may be a long way from application performance

Measurement Tools

- Benchmarks, Traces, Mixes
- Hardware:
  - Cost, delay, area, power estimation
- Simulation (many levels)
  - ISA, RT, Gate, Circuit
- Queuing Theory
- Rules of Thumb
- Fundamental “Laws”/Principles

Which has Higher Performance?

Plane	DC to Paris	Speed	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

- Time to run the task (Execution Time)
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns … (Performance)
  - Throughput, bandwidth

Performance Definition

- Performance is in units of things per sec (bigger is better)
- If we are primarily concerned with response time

$$\text{Performance (X)} = \frac{1}{\text{Execution\_time (X)}}$$

" X is n times faster than Y" **means**

$$n = \frac{\text{Performance (X)}}{\text{Performance (Y)}} = \frac{\text{Execution\_time (Y)}}{\text{Execution\_time (X)}}$$

- Speed of Concorde vs. Boeing 747
- Throughput of Boeing 747 vs. Concorde

Performance: Example

- **Time of Concorde vs. Boeing 747?**
  - Concorde is 1350 mph / 610 mph = 2.2 times faster
    = 6.5 hours / 3 hours

- **Throughput of Concorde vs. Boeing 747 ?**
  - Concorde is 178,200 pmph / 286,700 pmph = 0.62 "times faster"
  - Boeing is 286,700 pmph / 178,200 pmph = 1.6 "times faster"

- Boeing is 1.6 times ("60%") faster in terms of throughput
- Concorde is 2.2 times ("120%") faster in terms of flying time

- **We will focus primarily on execution time for a single job**

Amdahl's Law

- The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl's Law.
- Amdahl's Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.
- Amdahl's Law defines the speedup that can be gained by using a particular feature.

Amdahl's Law

- What is speedup?
- Suppose that we can make an enhancement to a computer that will improve performance when it is used.
- Speedup is the ratio

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

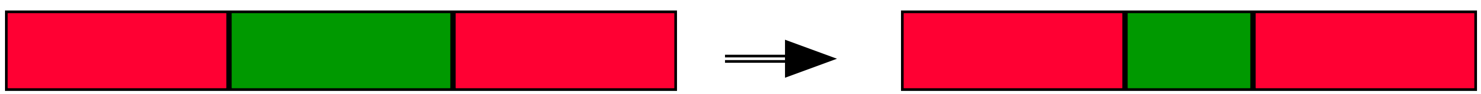
- Alternatively

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

Amdahl's Law

Speedup due to enhancement E:

$$\text{Speedup (E)} = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected

ExTime(w/E) = (1-F) x ExTime(w/o E) + F/S x ExTime(w/E)

Amdahl's Law

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times \left[(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right]$$

$$\text{Speedup}_{overall} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

Amdahl's Law: Example 1

Suppose that Floating point instructions are improved to run 2X; but only 10% of actual instructions are FP. What's the overall speedup gained?

F = 0.1 **S = 2**

$\text{ExTime}_{\text{new}}$ =

$\text{Speedup}_{\text{overall}}$ =

Amdahl's Law: Example 1

Suppose that Floating point instructions are improved to run 2X; but only 10% of actual instructions are FP. What's the overall speedup gained?

F = 0.1 **S = 2**

$$ExTime_{new} = ExTime_{old} \times (0.9 + 0.1/2) = 0.95 \times ExTime_{old}$$

$$Speedup_{overall} = \frac{1}{0.95} = 1.053$$

Amdahl's Law: Example 2

Suppose that we are considering an enhancement to the processor of a server system used for web serving. The new CPU is 10 times faster on computation in web serving application than the original processor. Assuming that the original CPU is busy with computation 40% of the time and is waiting for I/0 60% of time. What's the overall speedup gained by incorporating the enhancement?

F = 0.4 **S = 10**

Speedup(E) = 1/((1-0.4) + 0.4/10)
= 1.56

Amdahl's law: Example 2

Suppose we want to design a processor for graphic applications. Suppose that FP square root (FPSQR) computation is responsible for 20% of execution time and all FP instructions are responsible for 50% of execution time.

– Alternative 1: Enhance FPSQR hardware by a factor of 10.

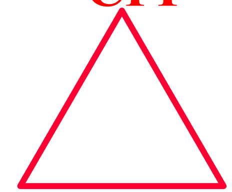
Speedup(FPSQR) = 1/((1-0.2)+ 0.2/10) = 1.22

– Alternative 2: Enhance the speedup of all FP operations by a factor of 1.6.

Speedup(FP) = 1/((1-0.5) + 0.5/1.6) = 1.28

Improving the performance of the FP operations overall is slightly better because of the higher frequency!

CPU Equation



$$\text{CPU time} = \frac{\#\,\text{Seconds}}{\text{Program}} = \frac{\#\,\text{Instructions}}{\text{Program}} \times \frac{\#\,\text{Cycles}}{\text{Instruction}} \times \frac{\#\,\text{Seconds}}{\text{Cycle}}$$

	Inst. Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

CPI: Cycles Per Instruction

- **Average Cycles per Instruction**

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times \text{I}_j$$

- **Instruction Frequency F_j**

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times \text{F}_j \qquad \text{where } \text{F}_j = \frac{\text{I}_j}{\text{Instruction Count}}$$

CPU Equation: Example 1

Reg-Reg Architecture

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	0.5	23%
Load	20%	5	1.0	45%
Store	10%	3	0.3	14%
Branch	20%	2	0.4	18%
			2.2	

Typical Mix

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?
- How does this compare with using branch prediction to save a cycle off the branch time?
- What if two ALU instructions could be executed at once?

CPU Equation: Example 2

- Suppose we have the following measurements:
  - Frequency of FP operations (others than FPSQR) = 25%
  - Average CPI of FP operations = 4
  - Average CPI of others instructions = 1.33
  - Frequency of FPSQR = 2%
  - CPI of FPSQR = 20
- Assume:
  - Alternative 1: decrease the CPI of FPSQR to 2
  - Alternative 2: decrease the average CPI of all operations to 2.5

Compare the two design alternatives using CPU performance equation.

CPU Equation: Example 2 (cont'd)

- CPI_original = 75% * 1.33 + 25% * 4 = 2
- CPI_newFPSQR = CPI_original – 2%*(CPI_oldFPSQR – CPI_newFPSQR)
  = 2 – 0.02*(20 – 2) = 1.64
- CPI_newFP = 75% * 1.33 + 25% * 2.5 = 1.625
- Speedup_newFP = CPU time original/CPU time new FP
  = CPU_original / CPI_newFP
  = 2.00/1.625 = 1.23
- Speedup_newFPSQR = 2.00 / 1.64 = 1.22

CPU Equation: Example 3

- Assume CPI = 1.0 ignoring branches (ideal)
- Assume branch solution was delaying for 3 cycles
- If 30% branch, delay 3 cycles on 30%

Op	**Freq**	**Cycles**	**CPI(i)**	**(% Time)**
Other	70%	1	0.7 (37%)	
Branch	30%	4	1.2 (63%)	

□ new CPI = 1.9

New machine is 1/1.9 = 0.52 times faster (i.e. slow!)

SPEC: System Performance Evaluation Cooperative

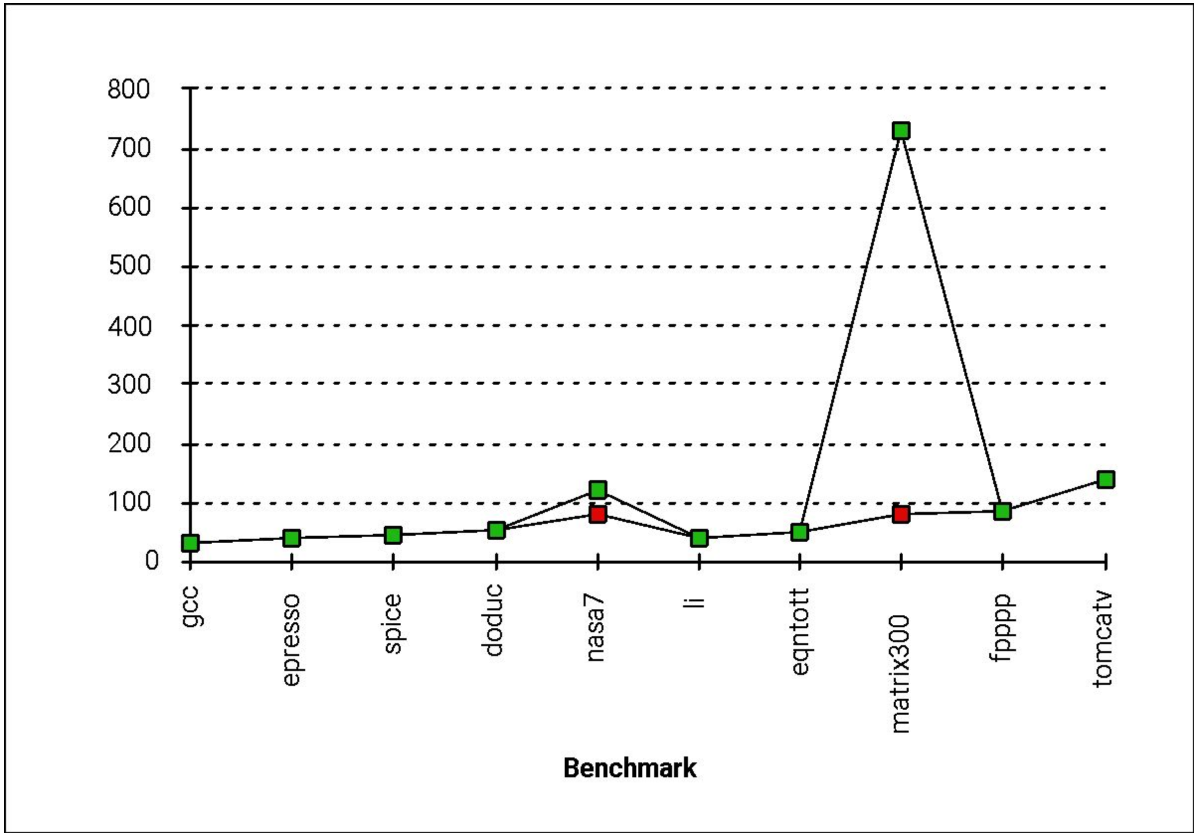
- First Round 1989
  - 10 programs yielding a single number ("SPECmarks")
- Second Round 1992
  - SPECInt92 (6 integer programs) and SPECfp92 (14 floating point programs)
    - Compiler Flags unlimited. March 93 of DEC 4000 Model 610:

```
  spice: unix.c:/def=(sysv,has_bcopy,"bcopy(a,b,c)=
memcpy(b,a,c)"
  wave5: /ali=(all,dcom=nat)/ag=a/ur=4/ur=200
  nasa7: /norecu/ag=a/ur=4/ur2=200/lc=blas
```

- Third Round 1995
  - new set of programs: SPECint95 (8 integer programs) and SPECfp95 (10 floating point)
  - "benchmarks useful for 3 years"
  - Single flag setting for all programs: SPECint_base95, SPECfp_base95

SPEC First Round

- One program: 99% of time in single line of code
- New front-end compiler could improve dramatically



More Performance Metrics

- Arithmetic mean (weighted arithmetic mean) tracks execution time: $\sum(T_i)/n$ or $\sum(W_i{*}T_i)$
- Harmonic mean (weighted harmonic mean) of rates (e.g., MFLOPS) tracks execution time: $n/\sum(1/R_i)$ or $n/\sum(W_i/R_i)$
- Normalized execution time is handy for scaling performance (e.g., X times faster than SPARCstation 10)
- But do not take the arithmetic mean of normalized execution time, use the geometric mean ($\prod(R_i)$^1/n)

Impact of Means on SPECmark89 for IBM 550

Program	Ratio to VAX ***Before***	***After***	Time ***Before***	***After***	Weighted Time ***Before***	***After***
gcc	30	29	49	51	8.91	9.22
espresso	35	34	65	67	7.64	7.86
spice	47	47	510	510	5.69	5.69
doduc	46	49	41	38	5.81	5.45
nasa7	**78**	**144**	**258**	**140**	**3.43**	**1.86**
li	34	34	183	183	7.86	7.86
eqntott	40	40	28	28	6.68	6.68
matrix300	**78**	**730**	**58**	**6**	**3.43**	**0.37**
fpppp	90	87	34	35	2.97	3.07
tomcatv	33	138	20	19	2.01	1.94
Mean	**54**	**72**	**124**	**108**	**54.42**	**49.99**

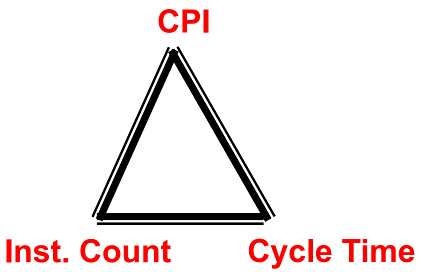
Geometric Ratio 1.33 *Arithmetic Ratio 1.16* *Weighted Arith. Ratio 1.09*

SPEC Performance: Summary

- “For better or worse, benchmarks shape a field”
- Good products created when have:
  - Good benchmarks
  - Good ways to summarize performance
- Given sales is a function in part of performance relative to competition, investment in improving product as reported by performance summary
- If benchmarks/summary inadequate, then choose between improving product for real programs vs. improving product to get more sales;
  Sales almost always wins!
- Execution time is the measure of computer performance!

Performance Metrics: Summary

- Design-time metrics:
  - Can it be implemented, in how long, at what cost?
  - Can it be programmed? Ease of compilation?
- Static metrics:
  - How many bytes does the program occupy in memory?
- Dynamic metrics:
  - How many instructions are executed?
  - How many bytes does the processor fetch to execute the program?
  - How many clocks are required per instruction?
  - How "lean" a clock is practical?
- Best Metric: Time to execute the program!



- Depends on instructions set, processor organization, and compiler

Fallacies and Pitfalls

- All exponential laws must come to an end
  - Dennard scaling (constant power density)
    - Stopped by threshold voltage
  - Disk capacity
    - 30-100% per year to 5% per year
  - Moore's Law
    - Most visible with DRAM capacity
    - ITRS disbanded
    - Only four foundries left producing state-of-the-art logic chips
    - 11 nm, 3 nm might be the limit

Fallacies and Pitfalls

- Microprocessors are a silver bullet
  - Performance is now a programmer's burden
- Falling prey to Amdahl's Law
- A single point of failure
- Hardware enhancements that increase performance also improve energy efficiency, or are at worst energy neutral
- Benchmarks remain valid indefinitely
  - Compiler optimizations target benchmarks

Fallacies and Pitfalls

- The rated mean time to failure of disks is 1,200,000 hours or almost 140 years, so disks practically never fail
  - MTTF value from manufacturers assume regular replacement
- Peak performance tracks observed performance
- Fault detection can lower availability
  - Not all operations are needed for correct execution

A Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds,
  - Which computer is faster?
  - How much faster?

- We know that A is n times faster than B if

$$\frac{\text{performance of A}}{\text{performance of B}} = \frac{\text{execution\_time of B}}{\text{execution\_time of A}} = n$$

- The performance ratio n is 15/10 =1.5
- So A is 1.5 times (50%) faster than B

Ratios of Measure: Side Note

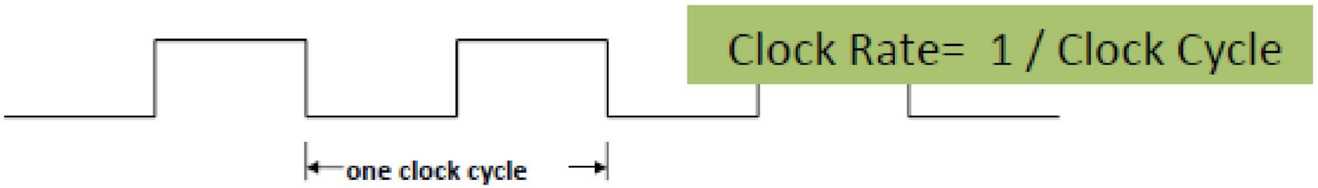
- For bigger-is-better metrics,
  - improved means increase
    - $V_{new} = 2.5 * V_{old}$
      - A metric increased by 2.5 times (sometimes written 2.5x)
      - A metric increased by 150% (x% increase == 0.01*x+1 times increase)
- For smaller-is-better metrics,
  - improved means decrease
    - e.g., Latency improved by 2x, means latency decreased by 2x (i.e., dropped by 50%)
    - e.g., Battery life worsened by 50%, means battery life decrease by 50%.

Examples

- Bigger-is-better examples
  - Bandwidth per dollar (e.g., in networking (GB/s)/$)
  - BW/Watt (e.g., in memory systems (GB/s)/W)
  - Work/Joule (e.g., instructions/joule)
  - In general
    - Multiply by big-is-better metrics, divide by smaller-is-better metrics
- Smaller-is-better examples
  - Cycles/Instruction (i.e., Time per work)
  - Latency * Energy -- Energy Delay Product
  - In general:
    - Multiply by smaller-is-better metrics, divide by bigger-is-better metrics

Clock Cycle and Clock Rate

- A clock cycle is a single electronic pulse of a CPU
  - To synchronize different parts of the circuit
  - To determine when events take place in the hardware
  - Processor runs at a constant clock rate
  - Clock cycle or tick or cycle = Discrete time interval
- Clock rate (frequency)
  - Number of clock cycles per second in hertz



- 1 nsec (10^{-9}) clock cycle => 1 GHz (10^{9}) clock rate
- 0.5 nsec clock cycle => 2 GHz clock rate

CPU Time (Execution Time)

- A program takes $15x10^{10}$ cycles to execute on a computer with a clock cycle time of 500 picosec.
  - How many seconds does it take for the program to execute?

$$\text{CPU Time} = \text{Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{Clock Cycles}}{\text{Clock Rate}}$$

- Clock Cycles:
  - How many cycles it takes for a program to execute!
- CPU Time (Execution time):
  - How many seconds it takes for a program to execute!

CPU Time Example

- Computer A has a 2 GHz clock rate, executes a program in 10 sec (CPU time)
- Designing Computer B by aiming for 6 sec CPU time
  - With a faster clock, but this causes 1.2 × clock cycles
- What is Computer B's clock rate?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A = 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

Comparing Computers

- Computers A and B implement the same ISA.
  - Computer A has a clock cycle time of 250 ps and an effective CPI of 2.0 for some program C
  - Computer B has a clock cycle time of 500 ps and an effective CPI of 1.2 for the same program.
- Which computer is faster and by how much?

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycles Time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

Comparing Computers

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$
$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

- Each computer executes the same number of instructions, I, so

$$\text{CPU time}_A = I \times 2.0 \times 250 \text{ ps} = 500 \times I\text{ps}$$
$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I\text{ps}$$

- Clearly, A is faster than B by the ratio of execution times

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = \frac{600 \text{ x } I\text{ps}}{500 \text{ x } I\text{ps}} = 1.2$$